Chapter 6

Parallel Paradigm for Temporal Association Rules

Temporal association rules are largely different from tradition association rules by the fact that temporal association rules attempt to model temporal relationships in the data. The present study proposes a novel methodology for extracting calendric association rules and hence the general rules for a timestamp transactional database. This chapter is organized as follows. Section 6.2 gives the description of proposed work for Parallel Computing of Compact Pattern Trees. Section 6.3 gives experimental results.

6.1 Introduction

Temporal association rule mining is first introduced by Wang, Yang and Muntz in years 1999-2001 together with the introduction of the TAR(Temporal Association Rule) algorithm [WYM99] as stated in [NHAF11]. Naqvi et al. stated in their paper that temporal association rule mining has been introduced in order to solve the problem on handling time-series by including time expression into association rules [GNTA10]. Temporal association helps to find the valuable relationship among the different itemsets, in temporal database. There are different types of temporal association rules defined in the literature such as intertransaction rules, episode rules, trend dependencies, sequence association rules and calendric association rules. In this chapter we experimented with the temporal database in order to describe real-life concepts such as the business in the first week of month and the remaining weeks of the same month is not constant. Data Mining Techniques are being evolved and enhanced over the years, to cater the needs of
handling massive data, to employ HPC (High Performance Computing) environment for building effective as well as timely advisory systems/agents. Later years, based on this data mining task the research activity in this area extended tremendously with new scalable data mining algorithms for various applications.

As stated in [ZHL01] existing association rule mining algorithms are not sufficient for extremely large datasets and new solutions still have to be found. In particular there is a need for algorithms that do not depend on high computation and repeated I/O scans. Parallelization is a viable solution. However, distributing and balancing the mining tasks between the processors without jeopardizing the global solution is not trivial. Several researchers [JX08], [ASP96], [ZOP96] employed parallel algorithms for association rule mining. FP-Forest [JX08] is to store data using multi-trees. Agrawal et al. [ASP96] first put forward three parallel algorithms, count distribution, data distribution and candidate distribution. Zaki et al. [ZOP96] proposed the Common Candidate Partitioned Database (CCPD) and the Partition Candidate Common Database (PCCD) algorithms, both algorithms are Apriori-based framework. As stated in [KK06] the computational cost of association rules mining can be reduced in four ways:

- By reducing the number of passes over the database
- By sampling the database
- By adding extra constraints on the structure of patterns
- Through parallelization.

This chapter in contrast to the above develops parallel schemes for aiming at extraction of calendar association rules. Zaki stated that association rule discovery techniques have gradually been adapted to parallel systems in order to take advantage of the higher speed
and greater storage capacity [Z99]. The transition to a distributed memory system requires the partitioning of the database among the processors, a procedure that is generally carried out indiscriminately.

In this chapter, the parallel algorithm proposed by Tanbeer et al. [TAJL08b] for Compact Pattern Trees has been modified to elicit incidental periodic knowledge and to discover seasonal trends which focus on business and decision perspectives in the context of temporal mining.

### 6.2 Parallel Construction of Compact Pattern Trees

In this chapter we modified the proposed parallel algorithm by Tanbeer et al. [TAJL09] in order to obtain periodic knowledge for databases having timestamp information and to discover the patterns within the same time or among different time sequences. Consider the transactional database shown in Table 6.1. A set of time series in the database indicate the transaction records from first week to last week in the month. In the following, we use the time-variant transactions for illustrations. In this application, we would like to mine the transaction database for a periodic knowledge. Considering TIDs as monthly transactions and each window are the week logs and mining the rules for that particular log in order to obtain knowledge for time series analysis.

The following is the procedure to discover periodic knowledge from Parallel Compact Pattern Tree.

- **Step 1:** Based on the application, database is split into required time intervals (like monthly transactions, weekly transactions etc.) by preprocessing using the timestamp information provided. This process of splitting database is called
sharding [LWZZC08]. This division of data is nothing but the fragmenting the dataset.

- **Step 2**: All fragments are parallely deployed to different processors.
- **Step 3(i)**: Each processor parallely constructs Transaction Tree (TT) for fragment dataset by maintaining item list occurrences (TL). Parallel Processing can occur on a multiprocessor computer or on a network of workstations or PCs.
- **Step 3(ii)**: Sorting TL in frequency descending order and maintaining as item sorted list (TS).
- **Step 3(iii)**: Based on TS, restructuring the TT and maintained as Restructured Tree (RT).
- **Step 3(iv)**: Each processor is having TT for fragment dataset.
- **Step 3(v)**: Then the mining process is carried out on temporal database (fragment dataset) by applying the algorithm FP-growth [HPY00]. Then the frequent patterns are generated by satisfying the predefined minimum support to each fragment dataset. Temporal Association rules are generated by satisfying the predefined minimum confidence.

From Step 3(i) to 3(v) the process executes parallely in different processors (control parallelism). Each processor generates CPTree in order to derive the periodic knowledge for fragment dataset such as weekly transactions.

The Transaction Tree and Restructure Tree construction are carried out using the OpenMP (Multi-processing tool) mechanism.

### 6.2.1 Illustrations of Construction and Restructure of CPTree

Compact Pattern Tree [TAJL08] requires only one scan. Assuming Table 6.1 as monthly transactional database and treating each ‘\( P_i \)’ (generated by applying the algorithm 6.2) as weekly transactions and below are the illustrations shown in figures. All four
windows database executes independently in different processors parallely. We can obtain periodic knowledge parallely at the same time. Considering P1 as first week timestamp dataset the construction and restructuring of Compact Pattern Tree is followed based on the algorithms 6.3 and 6.4 are shown in Figures 6.1 and 6.2 along with TL and TS. The same procedure executes independently for each window in each processor at the same time. The resultant CPTrees are generated by applying the algorithm 6.1 shown in Figures 6.4, 6.6 and 6.8 respectively.

**Table 6.1**

Sample Transaction Database used in the experiment

<table>
<thead>
<tr>
<th>Database</th>
<th>Date</th>
<th>TID</th>
<th>Item set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Week</td>
<td></td>
<td>t1</td>
<td>{a,b,d,g,e,c}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t2</td>
<td>{c,a,e}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t3</td>
<td>{d,f,b,a,e}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t4</td>
<td>{b,d,f}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t5</td>
<td>{a}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t6</td>
<td>{a,b,d}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t7</td>
<td>{d,a,b}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t8</td>
<td>{b,d,e}</td>
</tr>
<tr>
<td>2nd Week</td>
<td></td>
<td>t9</td>
<td>{a,c,b}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t10</td>
<td>{f}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t11</td>
<td>{c,b,a,e}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t12</td>
<td>{g}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t13</td>
<td>{a,b,c}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t14</td>
<td>{f,e}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t15</td>
<td>{a,b,c}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t16</td>
<td>{b,d,e}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t17</td>
<td>{b,d,e}</td>
</tr>
<tr>
<td>3rd Week</td>
<td></td>
<td>t18</td>
<td>{c,a,e}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t19</td>
<td>{b,d,f}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t20</td>
<td>{a,b,d}</td>
</tr>
</tbody>
</table>
Figure 6.1 first week Transactional Tree

Figure 6.2 first week Restructured Tree according to TS of Fig 6.1

Figure 6.3 second week Transactional Tree

Figure 6.4 second week Restructured Tree according to TS of Fig 6.3
• From Figures 6.2 and Figure 6.4 periodic knowledge can be derived. Then the mining process is carried out on temporal database (fragment dataset) by applying the algorithm FP-growth[HPY00]. Then the frequent patterns are generated by satisfying the predefined minimum support to each fragment dataset. Temporal Association rules are generated by satisfying the predefined minimum confidence.

6.2.1 Periodic Knowledge

Our approach is different from Tanbeer et al.[TAJ09] approach. Tanbeer proposed by dividing the whole database into partitions and constructing the tree in each site by maintaining header table list. Combining all the header table list and sorting in frequency descending order and named as HTA. With the help of HTA restructuring takes place parallely. Based on Tanbeer’s approach, we can derive only the global frequent patterns. Business process require several layers of knowledge description such as trends of purchasing (like weekly, monthly etc.,) So the developed framework is useful in such scenarios to derive knowledge suitable to user requirements. Further the proposed method is also useful to derive global frequent patterns and the additional data of subsequent periods (months) can be added to the respective processors. The incremental rule discovery method explained in Chapter 5 can be employed for updating periodic knowledge.

Algorithm 6.1 Parallel Compact Pattern Tree

**Input:** Transactional Database(D)

**Output:** TR₁,TR₂,TR₃,..............TRₙ

**Method:**

1. [WD₁,WD₂,........WDₙ]= Shard(D)
2. Invoke in parallel for i=1 to n
3. do
   a. TTᵢ= TransactionTree(WDᵢ)
   b. TRᵢ= Restructure Tree(TTᵢ)
4. done
Figure 6.5 third week Transactional Tree

Figure 6.6 third week Restructured Tree according to TS of fig 6.5

Figure 6.7 fourth week Transactional Tree

Figure 6.8 fourth week Restructured Tree according to TS of fig 6.7
Algorithm 6.2 Sharding

**Input:** Transactional Database(D)

**Output:** WD<sub>1</sub>, WD<sub>2</sub>, …… WD<sub>n</sub>

**Method:**

1. Divide D into various ‘n’ timestamps as WD<sub>1</sub> to WD<sub>n</sub>.

Algorithm 6.3 Constructing Transactional Tree

**Input:** WD: Periodic Data

N: Number of transactions in WD

**Output:** TT: Transactional Tree

TS: Item Sorted List

**Method:**

1. Create the root of a TT and label it as “null”
2. Read WD, N and set i=1
3. while (j<=N){
4.   Scan the transaction t<sub>j</sub> once
5.   Let the transaction be [p/P], where p is the first element and P is the remaining elements in the transaction, then call construct_tree ([p/P],r)
6.   j=j+1 }


(7) Find occurrences of each item in the tree and arrange each item based on occurrence. The arranged items are sorted in TS.

Function *construct_tree*

Input: \([p/P]\): p current element of a transaction, \(P\): Remaining portion of transaction  
\(T\): Current node of the Tree.

Method:

(6) If \(T\) has a child \(C\) such that \(C.item\_name=P.item\_name\)

(7) then increment count by 1.

(8) If length of \(P\)>0 and \(P\) is \([q/Q]\)

(9) then

a. *construct_tree* \(([q/Q],C)\)

(10) else

a. Attach \([p/P]\) as the child branch of \(T\).  

________________________________________________________

**Algorithm 6.4 Restructuring the Transactional Tree**

**Input:** TT: Transactional Tree  
TS: Item Sorted List

**Output:** TR: Compact Pattern Tree

**Method:**

(1) For each branch \(b_i\) in TT

(2) do
6.3 Experimentation and Analysis

The implementation of the existing and proposed algorithms is done using Java Programming Language. The construction of Transactional Tree and Restructure Tree are carried out using the OpenMP (Multi-processing tool) mechanism. Tests are performed using different size datasets. Different libraries like JOMP and Xstream Libraries are used. Performance is evaluated using MushRoom Data Set and synthetic dataset by assuming the temporal field as timestamp and the data is converted into transactional database suitable to our experiments. Table 6.2 describes the experimental results by using number of processors. Figure 6.9 shows the execution time of Parallel Compact Pattern Tree.
Table 6.2

The following table shows the mining time with single and multiple processors

*ms – milliseconds  * S-Single Processor  *P-Number of Processors

<table>
<thead>
<tr>
<th>Transactional Database (D)</th>
<th>Transaction Tree Time(*ms)</th>
<th>Restructure Time(*ms)</th>
<th>Mining Time(*ms)</th>
<th>Total time(*ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*S  *P</td>
<td>*S  *P</td>
<td>*S  *P</td>
<td>*S  *P</td>
</tr>
<tr>
<td>400</td>
<td>15  10</td>
<td>26  14</td>
<td>64  55</td>
<td>105 79</td>
</tr>
<tr>
<td>800</td>
<td>21  12</td>
<td>40  16</td>
<td>66  61</td>
<td>127 89</td>
</tr>
<tr>
<td>1000</td>
<td>22  14</td>
<td>42  22</td>
<td>67  63</td>
<td>131 99</td>
</tr>
<tr>
<td>2000</td>
<td>26  19</td>
<td>51  24</td>
<td>76  72</td>
<td>153 115</td>
</tr>
</tbody>
</table>

Figure 6.9  Execution time for Temporal Compact Pattern Tree
Dotted lines indicates number of transactions that are executed in single processor environment. Straight line indicates number of transactions that are executed in parallel environment.